Parallel Algorithm for Efficient Calculation of Second Derivatives of Conformational Energy Function in Internal Coordinates

SHUGO NAKAMURA, MITSUNORI IKEGUCHI, KENTARO SHIMIZU

Department of Biotechnology, The University of Tokyo, Yayoi 1-1-1, Bunkyo-ku, Tokyo 113-8657, Japan

Received 14 April 1998; accepted 30 June 1998

ABSTRACT: A parallel algorithm for efficient calculation of the second derivatives (Hessian) of the conformational energy in internal coordinates is proposed. This parallel algorithm is based on the master/slave model. A master processor distributes the calculations of components of the Hessian to one or more slave processors that, after finishing their calculations, send the results to the master processor that assembles all the components of the Hessian. Our previously developed molecular analysis system for conformational energy optimization, normal mode analysis, and Monte Carlo simulation for internal coordinates is extended to use this parallel algorithm for Hessian calculation on a massively parallel computer. The implementation of our algorithm uses the message passing interface and works effectively on both distributed-memory parallel computers and shared-memory parallel computers. We applied this system to the Newton–Raphson energy optimization of the structures of glutaminyl transfer RNA (Gln-tRNA) with 74 nucleotides and glutaminyl-tRNA synthetase (GlnRS) with 540 residues to analyze the performance of our system. The parallel speedups for the Hessian calculation were 6.8 for Gln-tRNA with 24 processors and 11.2 for GlnRS with 54 processors. The parallel speedups for the Newton–Raphson optimization were 6.3 for Gln-tRNA with 30 processors and 12.0 for GlnRS with 62 processors. J Comput Chem 19: 1716–1723, 1998

Keywords: internal coordinates; Hessian; parallel algorithm; message passing interface; Newton–Raphson optimization

Correspondence to: S. Nakamura; e-mail: shugo@bi.a.u-tokyo.ac.jp

CCC 0192-8651 / 98 / 151716-08

Introduction

The calculation of conformational energy and its derivatives is one of the most important computation steps in molecular simulations. The second derivative of the conformational energy function, the Hessian, is often required for simulations of biomolecules. Monte Carlo (MC) simulation using the Hessian of the target molecule was developed for the purpose of efficient sampling.[1] In this method, conformations of molecules are generated in each MC step that are biased toward the directions calculated from eigenvectors of the Hessian. The Newton–Raphson method and other methods for conformational energy optimization of biomolecules[2,3] also include the calculation of the Hessian in their algorithms. Normal mode analysis[4–6] requires energy optimization until all the eigenvalues of the Hessian become positive (i.e., the Hessian becomes positive definite), followed by the calculation of a generalized eigenproblem including the Hessian. In these simulations, the calculation of the Hessian is repeated so many times that an algorithm for calculating it rapidly is expected to greatly improve their performance.

To represent the conformations of proteins and nucleic acid molecules, two different coordinate systems are often used, Cartesian and dihedral angle coordinates (internal coordinates). In the latter, the degree of freedom can be reduced and one can save a lot of computation time and computer resources for a simulation. In the case of large molecules, however, calculating the Hessian is much more expensive than calculating conformational energy and its first derivative (gradient), even in internal coordinates. Thus, several algorithms for efficient calculations of the Hessian have been developed. One of the most efficient algorithms is the algorithm developed by Gō et al.[7–9] Their algorithm reduced the computation cost from about n^4 to n^2 (n is the degree of freedom) and is extended to handle multimolecular systems[10,11] and loop structures such as ribose rings.[12,13] However, when the target molecules have several hundred residues, the computation cost for calculating the Hessian is high even with their algorithm.

The recent progress in computer technology has made parallel computers widely available. When one can divide a problem into subproblems that can be solved independently, one can reduce computation cost and workspaces for calculations per processor by solving the problem on parallel computers, which leads to rapid calculation of large target molecules. Parallel algorithms for calculating the conformational energy and its first derivative in Cartesian coordinates and in internal coordinates have been proposed.[14–18] For molecular mechanics in internal coordinates, Totrov and Abagyan proposed parallel algorithms on shared-memory parallel computers for calculating the conformational energy, gradient, solvent accessible surface area, and electrostatic polarization energy, and for updating lists of interactions.[18] However, no parallel algorithm for calculating the Hessian in internal coordinates, which has a higher computation cost than those of the conformational energy and gradient, has yet been developed. In addition, no parallel algorithms for the gradient and the Hessian in internal coordinates have yet been developed that can be used on distributed-memory parallel computers.

In this article we propose an efficient parallel algorithm for calculating the Hessian in internal coordinates based on the algorithm of Gō et al. Our previously developed molecular analysis system for conformational energy optimization,[2] normal mode analysis,[4] and MC simulation for internal coordinates are extended to use this parallel algorithm for the Hessian calculation on a massively parallel computer. We applied this system to Newton–Raphson energy optimization of the structure of glutaminyl transfer RNA (Gln-tRNA) with 74 nucleotides and glutaminyl-tRNA synthetase (GlnRS) with 540 residues and analyzed the performance of our system.

Methods

PARALLEL ALGORITHM FOR HESSIAN

We represent the structure of a biomolecule in internal coordinates by rigid ''units'' (a group of one or more atoms) connected by rotatable ''bonds,'' as shown in Figure 1. We define the following quantities for easier description of the algorithm of Wako and Gō, according to ref. 9: (a, b) is the bond pair of a and b, which are the ath and bth bonds, respectively; a^* and b^* are the outside units of bonds a and b, respectively; a' and b' are the outside bonds of units a^* and b^*, respectively; $k(a, b)$ is the distance between bonds a and b, which is defined as the number of units between them; and k_{max} is the maximum value of

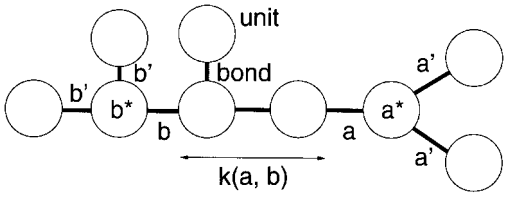

FIGURE 1. Schematic representation of units and bonds.

k appearing in the molecule. The condition $0 \leq k(a, b) \leq k_{max}$ holds for all the bond pairs of (a, b).

The algorithm of Gō et al. is briefly shown in Figure 2 (see refs. 7–9 for details). This algorithm reduces the computation cost from about n^4 to n^2 (n is the degree of freedom). Each component of the Hessian

$$\text{Hess}(a, b) \equiv \frac{\partial^2 E}{\partial\theta_a \partial\theta_b}$$

(E is the conformational energy function and θ is the internal coordinate corresponding to a bond) is calculated recurrently from a matrix R_{ab}, which is defined by

$$R_{ab} = l_{a^*b^*} + \sum_{a'} R_{a'b} + \sum_{b'} R_{ab'} - \sum_{a', b'} R_{a'b'}, \quad (1)$$

where $l_{a^*b^*}$ is a 6×6 nonsymmetric matrix whose components represent all the interactions between atoms in a^* and b^*.

When $k(a, b)$ is equal to a value of k, the following equations are always true:

$$k(a', b) = k(a, b') = k + 1, \quad (2)$$

$$k(a', b') = k + 2. \quad (3)$$

```
for (k = kmax ; k >= 0; k--)
    for ((a,b) such that k(a,b)==k)
        Calculate l a*b*
        Calculate R ab
          Rab= la*b* +
               Σa' Ra'b +Σb' Rab' - Σa'b' Ra'b'
        Calculate Hess ab
```

FIGURE 2. Description of the algorithm for the Hessian proposed by Gō et al.

From the above equations, the calculation of Hess(a, b) of which a and b are separated by k units [i.e., $k(a, b) = k$] only requires matrices R corresponding to $k + 1$ and $k + 2$. This helps to reduce the computation cost and the usage of computer resources. In the first step of the algorithm (Fig. 2), all Hess(a, b) whose $k(a, b)$ are equal to k_{max} are calculated. In the next step, Hess(a, b) whose $k(a, b)$ are equal to $k_{max} - 1$ are calculated. These calculations are repeated until $k(a, b)$ becomes zero.

Here we propose a parallel algorithm for the Hessian based on the above algorithm of Gō et al. Our implementation of this algorithm supports multimolecular systems and loop structures such as ribose rings by using the extension by Gō et al.[10–13] This algorithm is based on the master/slave model. A master processor distributes the calculations of components of the Hessian to one or more slave processors that, after finishing their calculations, send the results to the master processor that assembles all the components of the Hessian. Our algorithm is shown in Figure 3. Because all the calculations of Hess(a, b) whose $k(a, b)$ are equal to the same value can be executed independently, we divide and assign these calculations to different processors. For example, when

$$k(a_1, b_1) = k(a_2, b_2) = \ldots = k(a_m, b_m) = k, \quad (4)$$

the calculations of Hess(a_1, b_1), Hess$(a_2, b_2), \ldots,$ Hess(a_m, b_m) can be executed independently. So we assign the calculation of Hess$(a_1, b_1), \ldots,$ Hess$(a_{m/p}, b_{m/p})$ to the first processor, that of Hess$(a_{(m/p)+1}, b_{(m/p)+1}, \ldots,$ Hess$(a_{2m/p}, b_{2m/p})$ to the second processor and so on, where p is the number of available processors.

```
for (k = kmax ; k >= 0; k--)
    for ((a,b) such that k(a,b)==k && assigned)
        Calculate l a*b*
        Calculate R ab
          Rab= la*b* +
               Σa' Ra'b +Σb' Rab' - Σa'b' Ra'b'        (A)
        Calculate Hess ab                              (B)
    Unify Rab                                          (C)
Send Hess ab to Master                                 (D)
```

FIGURE 3. Our parallel algorithm for the Hessian.

Our algorithm consists of the following steps as shown in Figure 3.

1. Assign bond pairs (a, b) whose $k(a, b)$ are equal to k to the processors.
2. Calculate $l_{a^*b^*}$ and R_{ab} corresponding to an assigned bond pair of (a, b) (step A).
3. Calculate the component of the Hessian corresponding to the bond pair (step B).
4. Repeat steps A and B until all the processors have finished their assigned calculations.
5. Distribute the components of R_{ab} calculated by each processor to all the other processors (step C). We call this matrix-distribution procedure "unify."
6. Decrement k by one and return to step 1.
7. At the end of the calculations in the previous step $[k(a, b) = 0]$, send Hess(a, b) calculated by each processor to the master processor (step D), which then assembles all the components of the Hessian.

The algorithm of the unify procedure is shown in Figure 4. The purpose of this procedure is to distribute all the components of R_{ab} to each processor.

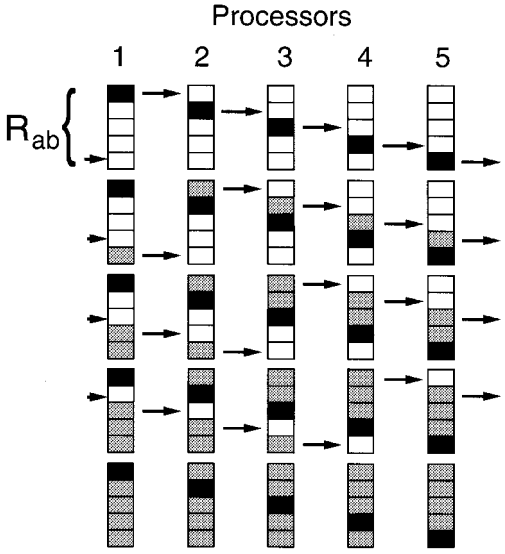

FIGURE 4. Algorithm of the unify procedure in the case of five processors. Black portions: the components of R_{ab} calculated by each processor. Gray portions: the components of R_{ab} sent by adjacent processors. Processor i sends the components of R_{ab} that are received from processor $(i - 1)$ in the previous step. After four steps, all of the processors have the whole of R_{ab}.

In the first step of the unify procedure, processor i sends the components of R_{ab} that are calculated in the current step (black portions in Fig. 4) to processor $(i + 1)$. In the next step, processor i sends the components of R_{ab} that are received from processor $(i - 1)$ in the previous step (gray portions in Fig. 4). After the components of R_{ab} have been sent $(p - 1)$ times, all the processors have the whole of R_{ab}.

PARALLEL ALGORITHM FOR NEWTON–RAPHSON OPTIMIZATION

We integrated our parallel algorithm for the Hessian calculation into our previously developed molecular analysis system for simulations in internal coordinates and estimated the improvement in total computation time for Newton–Raphson optimization on a parallel computer. We implemented our algorithm as a library written in C using the message passing interface (MPI) for the communication protocol, which is a programming environment widely used in scientific computing, so our algorithm works effectively on both distributed-memory parallel computers such as the Hitachi SR2201 and shared-memory parallel computers such as the Sun UltraEnterprise.

The calculations of the gradient and the conformational energy included in one step of the Newton–Raphson optimization are also parallelized. The parallel algorithm for the gradient is based on the algorithm of Wako and Gō.[9] In the same way as the Hessian, we divide and assign to different processors the calculations of components of the gradient. The computation cost for the gradient is much smaller than that of the Hessian. So in the Newton–Raphson optimization we assign only a few processors to calculate the gradient, and at the same time we calculate the Hessian with the rest of the processors.

The parallel algorithm for the conformational energy is rather simple; we divide and assign the calculations of the interactions between atom pairs to different processors.

TARGET MOLECULE

We executed the Newton–Raphson optimization of the structures of Gln-tRNA with 74 nucleotides and GlnRS with 540 residues and evaluated the performance of our algorithm. The degrees of freedom were 441 for Gln-tRNA and 2621 for GlnRS. The platform for our calculations was a Hitachi SR2201. Atom coordinates were

taken from the Brookhaven Protein Data Bank (entry name: 1GTR, A chain for GlnRS and B chain for Gln-tRNA).

Results and Discussion

PARALLEL SPEEDUPS FOR HESSIAN

The plots of computation time for calculating the Hessian against the number of processors for Gln-tRNA and GlnRS are shown in Figure 5. The parallel speedups for the Hessian calculation were 6.8 for Gln-tRNA with 24 processors and 11.2 for GlnRS with 54 processors.

Figure 6 shows N_{proc}/S plotted against $N_{\mathrm{proc}} - 1$ for GlnRS, where N_{proc} is the number of processors and S is speedup. According to Amdahl's law,[19] the speedup S is expressed as follows:

$$S = \frac{N_{\mathrm{proc}}}{1 + (N_{\mathrm{proc}} - 1)\alpha}, \qquad (5)$$

where α is the fraction of the serial bottleneck. In Figure 6 the α corresponds to the slope. We estimated α as 0.065, so the serial bottleneck is about 6.5% and the upper bound of the best speedup $(1/\alpha)$ is 15.36 in our implementation on the SR2201.

The performance was not significantly improved with about more than 24 processors for both GlnRS and Gln-tRNA. To clarify the reason for this, we analyzed the individual computation times for steps (A)–(D) in Figure 3 of the Hessian calculation. The results are shown in Tables I and II. For the calculation of $l_{a^*b^*}$, R_{ab} (A), and one component of the Hessian (B), the parallel speedups were almost 100%. But for the unify procedure (C), computation time increased with the number of processors. Computation time for the unify procedure (t_u) depends on the number of processors (p), the total amount of data corresponding to all the components of R_{ab} (D), and the time for setting up for communication (β). It is expressed by

$$t_u = d/p \cdot (p - 1) + \beta p. \qquad (6)$$

The amount of data to be sent to other processors in a unify step (i.e., the number of components of a submatrix of R_{ab}) decreases in inverse proportion to the number of processors (d/p), and the number of unify steps required for data distri-

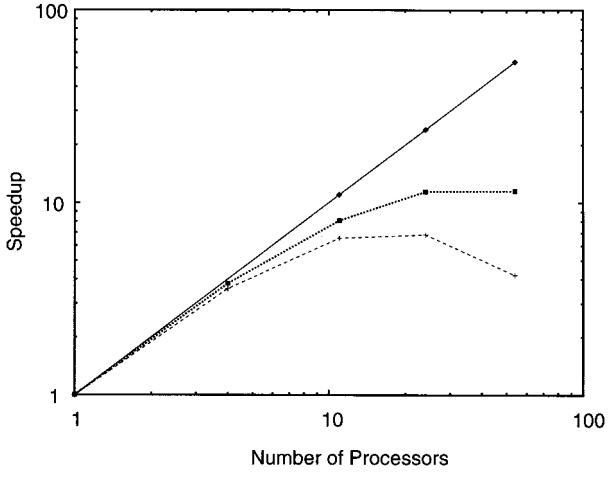

FIGURE 5. Speedups for calculating Hessians for Gln-tRNA and GlnRS plotted against the number of processors: (—) the ideal speedup, (- - -) the actual speedup for Gln-tRNA, and (· · ·) the actual speedup for GlnRS.

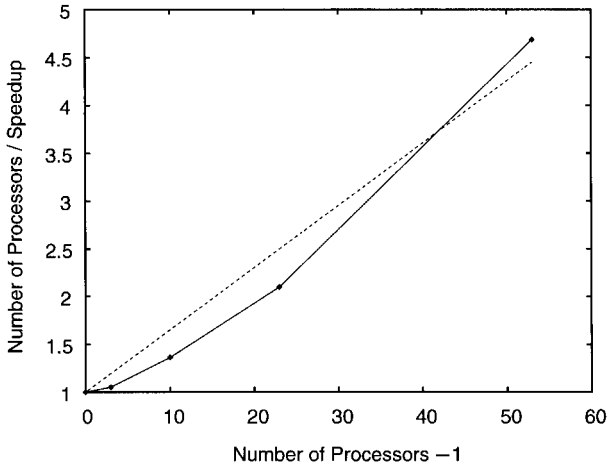

FIGURE 6. N_{proc}/S plotted against $N_{\mathrm{proc}} - 1$: (—) the actual plot for GlnRS and (· · ·) fitted line ($y = 0.065x + 1.0$). The slope in this plot corresponds to the fraction of the serial bottleneck according to Amdahl's law. Our fraction of the bottleneck is estimated as 6.5% and the upper bound of the best speedup $(1/\alpha)$ is 15.36 in our implementation on the SR2201.

TABLE I.
Computation Time (s) for Steps (A – D) of Hessian Calculation for Gln-tRNA.

	Number of Processors				
	1	4	11	24	54
A	13.95	3.49	1.35	0.64	0.32
B	0.55	0.14	0.06	0.02	0.01
C	0.00	0.54	0.96	1.59	3.25
D	0.01	0.04	0.04	0.04	0.05

TABLE II.
Computation Time (s) for Steps (A – D) of Hessian Calculation for GlnRS.

	Number of Processors				
	1	4	11	24	54
A	258.94	62.55	22.33	10.37	4.71
B	20.67	5.25	1.87	0.86	0.37
C	0.00	8.00	11.16	14.29	20.33
D	0.75	1.16	1.25	1.28	1.18

bution increased in proportion to the number of processors ($p - 1$). As a result, the total computation time for the unify is almost linear with respect to the number of processors (βp). The more the processors there were, the less significant the sum of the computation times for steps (A) and (B) became. And with more than about 24 processors, the computation time for the unify became dominant. Gln-tRNA has less degrees of freedom than GlnRS, so the ratio of the computation cost for the unify in the calculation became larger. Figures 7 and 8 show the number of bond pairs (a, b) plotted against the distance between bonds, $k(a, b)$, for Gln-tRNA and GlnRS, respectively. Figures 9 and 10 show the computation time plotted against $k(a, b)$ for Gln-tRNA and GlnRS, respectively. As the value of $k(a, b)$ decreased from $k_{\max}$ to 0, the number of bond pairs increased almost linearly (Figs. 7, 8). This caused the computation cost for the unify procedure to become large. This tendency increased with the number of processors (PEs), so the performance improvement deteriorated. This is clearly shown in Figures 9 and 10.

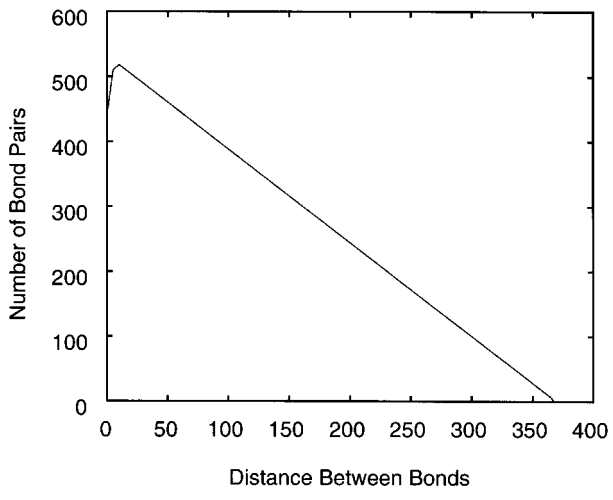

FIGURE 7. Number of bond pairs (a, b) plotted against the distance between bonds (k) for Gln-tRNA.

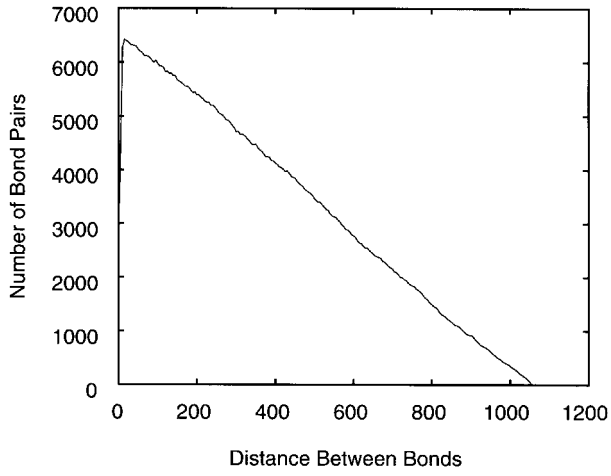

FIGURE 8. Number of bond pairs (a, b) plotted against the distance between bonds (k) for GlnRS.

To avoid this we plan to develop an algorithm in which we analyze the dependencies of the calculations of each component of R_{ab}, $R_{a'b}$, $R_{ab'}$, and $R_{a'b'}$, and assign them to processors such that the required number of communications is reduced. We are now designing a parallel programming environment that supports such processor assignment and we will evaluate the performance of this algorithm on the programming environment.

PARALLEL SPEEDUPS FOR NEWTON–RAPHSON OPTIMIZATION

The Newton–Raphson energy optimizations of Gln-tRNA and GlnRS with 1, 6, 14, 30, and 62 processors were executed and the computation

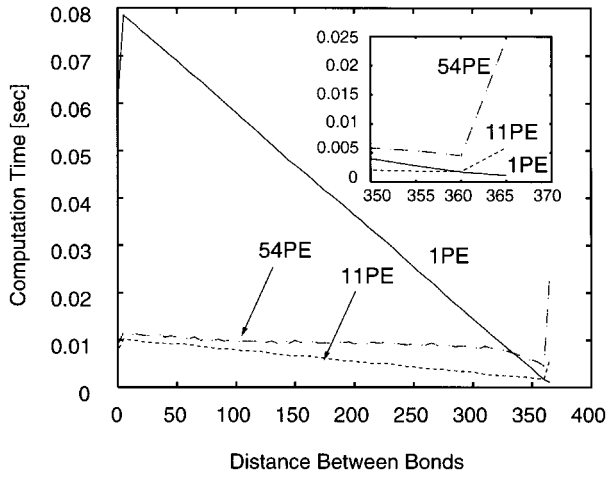

FIGURE 9. Computation time plotted against the distance between bonds (k) for Gln-tRNA.

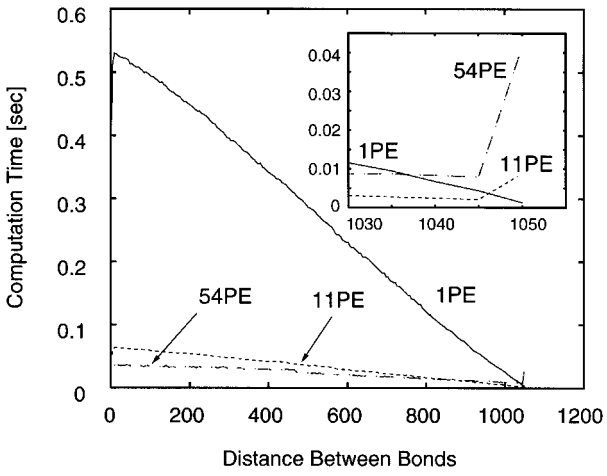

FIGURE 10. Computation time plotted against the distance between bonds (k) for GlnRS.

times were analyzed. We used 1, 2, 3, 6, and 8 processors for calculating the gradient; and 1, 4, 11, 24, and 54 processors for calculating the Hessian. The plots of computation time for one step of the Newton–Raphson optimization against the number of processors used for Gln-tRNA and GlnRS are shown in Figure 11. One step of the Newton–Raphson optimization included five evaluations of conformational energy function for Gln-tRNA and four for GlnRS.

For the calculation of the gradient and the conformational energy, we got almost 100% parallel speedups. The total parallel speedups of computation time for one step of the Newton–Raphson optimization were 6.3 for Gln-tRNA with 30 processors and 12.0 for GlnRS with 62 processors. The ratio of computation time for the calculation of the Hessian in one step of the Newton–Raphson optimization was about 40–56%. This means that the development of a better algorithm for the Hessian calculation may still improve the performance of the Newton–Raphson optimization.

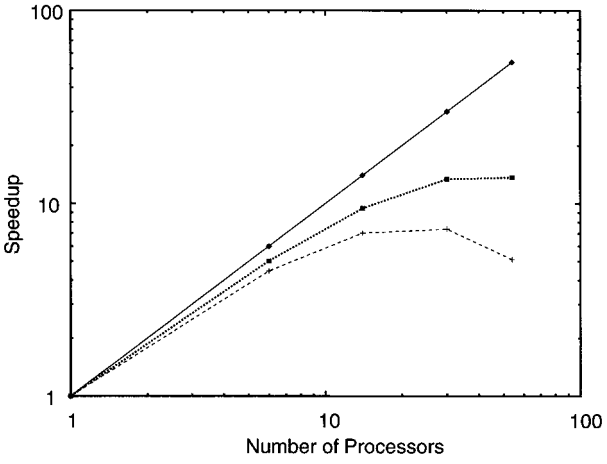

FIGURE 11. Speedups for calculating one step of the Newton–Raphson optimization for Gln-tRNA and GlnRS plotted against the number of processors: (—) the ideal speedup, (- - -) the actual speedup for Gln-tRNA, and (· · ·) the actual speedup for GlnRS.

Conclusion

We proposed an efficient parallel algorithm for the calculation of the Hessian in internal coordinates based on the algorithm of Gō et al. We extended our previously developed molecular analysis system for simulations of biomolecules in internal coordinates to use this parallel algorithm on a massively parallel computer and applied this system to the Newton–Raphson energy optimization of the structure of GlnRS. The parallel speedups for the Hessian calculation were 6.8 for Gln-tRNA with 24 processors and 11.2 for GlnRS with 54 processors. The parallel speedups for the Newton–Raphson optimization were 6.3 for Gln-tRNA with 30 processors and 12.0 for GlnRS with 62 processors. The decline of the performance was mainly caused by the unify procedure, and we are developing a new algorithm based on the dependence-driven model to omit this procedure.

Acknowledgment

We thank Dr. M. Tateno for providing us with the missing coordinates in the GlnRS and for helpful discussions.

References

1. T. Noguti and N. Gō, *Biopolymers*, **24**, 527 (1985).
2. S. Nakamura, H. Hirose, M. Ikeguchi and J. Doi, *J. Phys. Chem.*, **99**, 8374 (1995).
3. R. Abagyan, M. Totrov, and D. Kuznetsov, *J. Comput. Chem.*, **15**, 488 (1994).
4. S. Nakamura and J. Doi, *Nucl. Acids Res.*, **22**, 514 (1994).
5. N. Gō, T. Noguti, and T. Nishikawa, *Proc. Natl. Acad. Sci. U.S.A.*, **80**, 3696 (1983).
6. J.-F. Gibrat and N. Gō, *Proteins*, **8**, 258 (1990).
7. T. Noguti and N. Gō, *J. Phys. Soc. Jpn.*, **52**, 3685 (1983).
8. H. Abe, W. Braun, T. Noguti, and N. Gō, *Comput. Chem.*, **8**, 239 (1984).

9. H. Wako and N. Gō, *J. Comput. Chem.*, **8**, 625 (1987).
10. W. Braun, S. Yoshioki, and N. Gō, *J. Phys. Soc. Jpn.*, **53**, 3269 (1984).
11. J. Higo, Y. Seno, and N. Gō, *J. Phys. Soc. Jpn.*, **54**, 4053 (1985).
12. M. Tomimoto and N. Gō, *J. Phys. Chem.*, **99**, 563 (1995).
13. M. Tomimoto, H. Wako, and N. Gō, *J. Comput. Chem.*, **17**, 910 (1996).
14. J. E. Mertz, D. J. Tobias, C. L. Brooks III, and U. C. Singh, *J. Comput. Chem.*, **12**, 1270 (1991).
15. J. F. Janak and P. C. Pattnaik, *J. Comput. Chem.*, **13**, 533 (1992).
16. J. F. Janak and P. C. Pattnaik, *J. Comput. Chem.*, **13**, 1098 (1992).
17. S. E. De Bolt and P. A. Kollman, *J. Comput. Chem.*, **14**, 312 (1993).
18. M. Totrov and R. Abagyan, *J. Comput. Chem.*, **15**, 1105 (1994).
19. G. M. Amdahl, In *Proceedings of the AFIPS Conference*, AFIPS Press, Vol. 30, Atlantic City, NJ, 1967, p. 483.